Spam Filter Case Study

December 30, 2021

1 Pratical Exam On Spam Filtering

Name: Aditya C. Karpe Roll No: 166 Seat No: T-214161 Prn No: 02202002
Class: TY-Comp-SCET Div: C Batch: 3 Sem: 5th
Day : Wednesday Date: 29th Dec 2021
Email :aditya.karpe@mitaoe.ac.in Phone No: +91 7387624528

[1]:
```
import pandas as pd

sms_spam = pd.read_csv('SpamFiltering.csv', sep='\t', #  Reading csv &
 ↪converting into dataframe
header=None, names=['Label', 'SMS'])

print(sms_spam.shape)
sms_spam.head()       # Reading all dataset values
```

```
(5572, 2)
```

[1]:

	Label	SMS
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

[2]:
```
sms_spam['Label'].value_counts(normalize=True) # Counting the Label Values
```

[2]:
```
ham     0.865937
spam    0.134063
Name: Label, dtype: float64
```

[3]:
```
# Randomize the dataset
data_randomized = sms_spam.sample(frac=1, random_state=1)

# Calculate index for split
training_test_index = round(len(data_randomized) * 0.8)
```

```
# Split into training and test sets
training_set = data_randomized[:training_test_index].reset_index(drop=True)
test_set = data_randomized[training_test_index:].reset_index(drop=True)

print(training_set.shape) # Calculating no of rows and col for ham & spam
print(test_set.shape)
```

```
(4458, 2)
(1114, 2)
```

[4]:
```
training_set['Label'].value_counts(normalize=True) #frequency of training set
```

[4]:
```
ham     0.86541
spam    0.13459
Name: Label, dtype: float64
```

[5]:
```
test_set['Label'].value_counts(normalize=True) #frequency of test set
```

[5]:
```
ham     0.868043
spam    0.131957
Name: Label, dtype: float64
```

[6]:
```
# Before cleaning
training_set.head(3)
```

[6]:

	Label	SMS
0	ham	Yep, by the pretty sculpture
1	ham	Yes, princess. Are you going to make me moan?
2	ham	Welp apparently he retired

[7]:
```
# After cleaning
training_set['SMS'] = training_set['SMS'].str.replace(
   '\W', ' ') # Removes punctuation
training_set['SMS'] = training_set['SMS'].str.lower()
training_set.head(3) #converting into lower case
```

```
<ipython-input-7-82891ec54ccc>:2: FutureWarning: The default value of regex will
change from True to False in a future version.
  training_set['SMS'] = training_set['SMS'].str.replace(
```

[7]:

	Label	SMS
0	ham	yep by the pretty sculpture
1	ham	yes princess are you going to make me moan
2	ham	welp apparently he retired

[8]:
```
training_set['SMS'] = training_set['SMS'].str.split() #string split
```

```
vocabulary = [] #adding str in vocabulary one by one
for sms in training_set['SMS']:
    for word in sms:
        vocabulary.append(word) #append all word in array

vocabulary = list(set(vocabulary))
```

[9]:
```
len(vocabulary) #finding the len of the array
```

[9]:
```
7783
```

[10]:
```
word_counts_per_sms = {'secret': [2,1,1],
                       'prize': [2,0,1],
                       'claim': [1,0,1],
                       'now': [1,0,1],
                       'coming': [0,1,0],
                       'to': [0,1,0],
                       'my': [0,1,0],
                       'party': [0,1,0],
                       'winner': [0,0,1]
                      } # Counting the word/sms

word_counts = pd.DataFrame(word_counts_per_sms)
word_counts.head()
```

[10]:

	secret	prize	claim	now	coming	to	my	party	winner
0	2	2	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1	0
2	1	1	1	1	0	0	0	0	1

[11]:
```
word_counts_per_sms = {unique_word: [0] * len(training_set['SMS']) for␣
 ↪unique_word in vocabulary} #checking unique word from sms

for index, sms in enumerate(training_set['SMS']):   #multipling unique words in␣
 ↪vocabulary array with training set
    for word in sms:
        word_counts_per_sms[word][index] += 1
```

[12]:
```
word_counts = pd.DataFrame(word_counts_per_sms)  # converting output of␣
 ↪word_count/sms into dataframe
word_counts.head()
```

[12]:

	bsnl	rush	biola	now1	kanagu	ams	notifications	smear	qet	\
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	

```
4     0     0      0     0       0    0              0      0    0

   practicing  ...  09050001295  mahaveer  dan  varunnathu  placed  details  \
0           0  ...            0         0    0           0       0        0
1           0  ...            0         0    0           0       0        0
2           0  ...            0         0    0           0       0        0
3           0  ...            0         0    0           0       0        0
4           0  ...            0         0    0           0       0        0

   remembr  12mths  breather  marriage
0        0       0         0         0
1        0       0         0         0
2        0       0         0         0
3        0       0         0         0
4        0       0         0         0

[5 rows x 7783 columns]
```

[13]:
```
training_set_clean = pd.concat([training_set, word_counts], axis=1) # concat␣
↪word_set & traing set using axis 1
training_set_clean.head()
```

[13]:
```
  Label                                                SMS  bsnl  rush  biola  \
0   ham                  [yep, by, the, pretty, sculpture]     0     0      0
1   ham  [yes, princess, are, you, going, to, make, me,...     0     0      0
2   ham                    [welp, apparently, he, retired]     0     0      0
3   ham                                          [havent]     0     0      0
4   ham  [i, forgot, 2, ask, Ãij, all, smth, there, s, a,...     0     0      0

   now1  kanagu  ams  notifications  smear  ...  09050001295  mahaveer  dan  \
0     0       0    0              0      0  ...            0         0    0
1     0       0    0              0      0  ...            0         0    0
2     0       0    0              0      0  ...            0         0    0
3     0       0    0              0      0  ...            0         0    0
4     0       0    0              0      0  ...            0         0    0

   varunnathu  placed  details  remembr  12mths  breather  marriage
0           0       0        0        0       0         0         0
1           0       0        0        0       0         0         0
2           0       0        0        0       0         0         0
3           0       0        0        0       0         0         0
4           0       0        0        0       0         0         0

[5 rows x 7785 columns]
```

[14]:
```
# Isolating spam and ham messages first
spam_messages = training_set_clean[training_set_clean['Label'] == 'spam']
```

```
ham_messages = training_set_clean[training_set_clean['Label'] == 'ham']

# P(Spam) and P(Ham)
p_spam = len(spam_messages) / len(training_set_clean) #
p_ham = len(ham_messages) / len(training_set_clean)

# N_Spam
n_words_per_spam_message = spam_messages['SMS'].apply(len)
n_spam = n_words_per_spam_message.sum()
#NSpam is equal to the number of words in all the spam messages âĂŤ
#it's not equal to the number of spam messages, and it's not equal to the total␣
 ↪number of unique words in spam messages.

# N_Ham
n_words_per_ham_message = ham_messages['SMS'].apply(len) #formula
n_ham = n_words_per_ham_message.sum()
#NHam is equal to the number of words in all the non-spam messages âĂŤ
#it's not equal to the number of non-spam messages, and it's not equal to the␣
 ↪total number of unique words in non-spam messages.

# N_Vocabulary
n_vocabulary = len(vocabulary)

# Laplace smoothing
alpha = 1
```

[15]:
```
#Now that we have the constant terms calculated above, we can move on with␣
 ↪calculating the parameters P(wi|Spam) and P(wi|Ham).
#P(wi|Spam) and P(wi|Ham) will vary depending on the individual words. For␣
 ↪instance, P("secret"|Spam) will have a certain probability value,
#while P("cousin"|Spam) or P("lovely"|Spam) will most likely have other values.
#Therefore, each parameter will be a conditional probability value associated␣
 ↪with each word in the vocabulary.

# Initiate parameters
parameters_spam = {unique_word:0 for unique_word in vocabulary}
parameters_ham = {unique_word:0 for unique_word in vocabulary}

# Calculate parameters
for word in vocabulary:
    n_word_given_spam = spam_messages[word].sum() # spam_messages already defined
    p_word_given_spam = (n_word_given_spam + alpha) / (n_spam +␣
 ↪alpha*n_vocabulary)
    parameters_spam[word] = p_word_given_spam

    n_word_given_ham = ham_messages[word].sum() # ham_messages already defined
```

```
    p_word_given_ham = (n_word_given_ham + alpha) / (n_ham + alpha*n_vocabulary)
    parameters_ham[word] = p_word_given_ham
```

[16]:
```
#Let's start by writing a first version of this function. For the classify()␣
 ↪function below, notice that:

#The input variable message needs to be a string.
#We perform a bit of data cleaning on the string message.
#We remove the punctuation using the re.sub() function.
#We bring all letters to lower case using the str.lower() method.
#We split the string at the space character and transform it into a Python list␣
 ↪using the str.split() method.
#We calculate p_spam_given_message and p_ham_given_message.
#We compare p_spam_given_message with p_ham_given_message and then print a␣
 ↪classification label.

import re

def classify(message):
    '''
    message: a string
    '''

    message = re.sub('\W', ' ', message)
    message = message.lower().split()

    p_spam_given_message = p_spam
    p_ham_given_message = p_ham

    for word in message:
        if word in parameters_spam:
            p_spam_given_message *= parameters_spam[word]

        if word in parameters_ham:
            p_ham_given_message *= parameters_ham[word]

    print('P(Spam|message):', p_spam_given_message)
    print('P(Ham|message):', p_ham_given_message)

    if p_ham_given_message > p_spam_given_message:
        print('Label: Ham')
    elif p_ham_given_message < p_spam_given_message:
        print('Label: Spam')
    else:
        print('Equal proabilities, have a human classify this!')
```

[17]:
```
classify('WINNER!! This is the secret code to unlock the money: C3421.')
#We'll now test the spam filter on two new messages. One message is obviously␣
 ↪spam, and the other is obviously ham.
```

```
P(Spam|message): 1.3481290211300841e-25
P(Ham|message): 1.9368049028589875e-27
Label: Spam
```

[18]:
```
classify("Sounds good, Tom, then see u there")
```

```
P(Spam|message): 2.4372375665888117e-25
P(Ham|message): 3.687530435009238e-21
Label: Ham
```

[19]:
```
#The two results look promising, but we can  see how well the filter does on our␣
 ↪test set, which has 1,114 messages.
#We'll start by writing a function that returns classification labels instead of␣
 ↪printing them.

def classify_test_set(message):
    '''
    message: a string
    '''

    message = re.sub('\W', ' ', message)
    message = message.lower().split()

    p_spam_given_message = p_spam
    p_ham_given_message = p_ham

    for word in message:
        if word in parameters_spam:
            p_spam_given_message *= parameters_spam[word]

        if word in parameters_ham:
            p_ham_given_message *= parameters_ham[word]

    if p_ham_given_message > p_spam_given_message:
        return 'ham'
    elif p_spam_given_message > p_ham_given_message:
        return 'spam'
    else:
        return 'needs human classification'
```

[20]:
```
test_set['predicted'] = test_set['SMS'].apply(classify_test_set)
test_set.head()
```

```
# we have a function that returns labels instead of printing them, we can use it␣
 ↪to create a new column in our test set.
```

[20]:

	Label	SMS	predicted
0	ham	Later i guess. I needa do mcat study too.	ham
1	ham	But i haf enuff space got like 4 mb...	ham
2	spam	Had your mobile 10 mths? Update to latest Oran...	spam
3	ham	All sounds good. Fingers . Makes it difficult ...	ham
4	ham	All done, all handed in. Don't know if mega sh...	ham

[21]:
```
#We can compare the predicted values with the actual values to measure how good␣
 ↪our spam filter is with classifying new messages.
#To make the measurement, we'll use accuracy as a metric.
correct = 0
total = test_set.shape[0]

for row in test_set.iterrows():
    row = row[1]
    if row['Label'] == row['predicted']:
        correct += 1

print('Correct:', correct)
print('Incorrect:', total - correct)
print('Accuracy:', correct/total)
```

```
Correct: 1100
Incorrect: 14
Accuracy: 0.9874326750448833
```